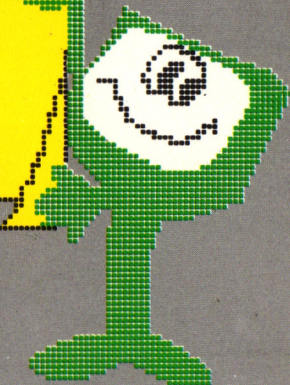
VIDEO
BASIC
20 VIDEOLEZIONI DI BASIC
PER IMPARARE CON L'MSX

GRUPPO
EDITORIALE
JACKSON

CPU: organizzazione esterna e interna
I BUS
Il linguaggio macchina
Vantaggi e svantaggi dell'Assembler
DEFUSR, USR, CALL VARPTR, INP, OUT, WAIT
Memorizzazione di programmi in linguaggio macchina
Videoesercizi
Videogioco n° 18

18
MSX
Per tutti i sistemi MSX

VIDEOBASIC MSX
Pubblicazione quattordicinale
edita dal Gruppo Editoriale Jackson
Direttore Responsabile:
Giampietro Zanga
Direttore e Coordinatore
Editoriale: Roberto Pancaldi
Autore: Softidea -
Via Indipendenza 88-90 - Como
Redazione software:
Michele Casertelli
Francesco Franceschini
Progetto grafico:
Studio Nuovidea - via Longhi, 16 - Milano
Impaginazione:
Moreno Confalone
Illustrazioni:
Cinzia Ferrari, Silvano Scolari
Fotografie:
Marcello Longhini
Distribuzione: SODIP
Via Zuretti, 12 - Milano
Fotocomposizione: Lineacomp S.r.l.
Via Rosellini, 12 - Milano
Stampa: Grafika '78
Via Trieste, 20 - Pioltello (MI)
Direzione e Redazione:
Via Rosellini, 12 - 20124 Milano
Tel. 02/6880951/5

Autorizzazione alla pubblicazione Tribunale di Milano n° 422 del 22-9-1984
Spedizione in abbonamento postale Gruppo II/70 (autorizzazione della Direzione Provinciale delle PPTT di Milano).
Prezzo del fascicolo L. 8.000
Abbonamento comprensivo di 5 raccoglitori L. 165.000
I versamenti vanno indirizzati a: Gruppo Editoriale Jackson S.p.A. - Via Rosellini, 12 20124 Milano, mediante emissione di assegno bancario o cartolina vaglia oppure utilizzando il c.c.p. n° 11666203.
I numeri arretrati possono essere richiesti direttamente all'editore inviando L. 10.000 cdu. mediante assegno bancario o vaglia postale o francobolli.
Non vengono effettuate spedizioni contrassegno.

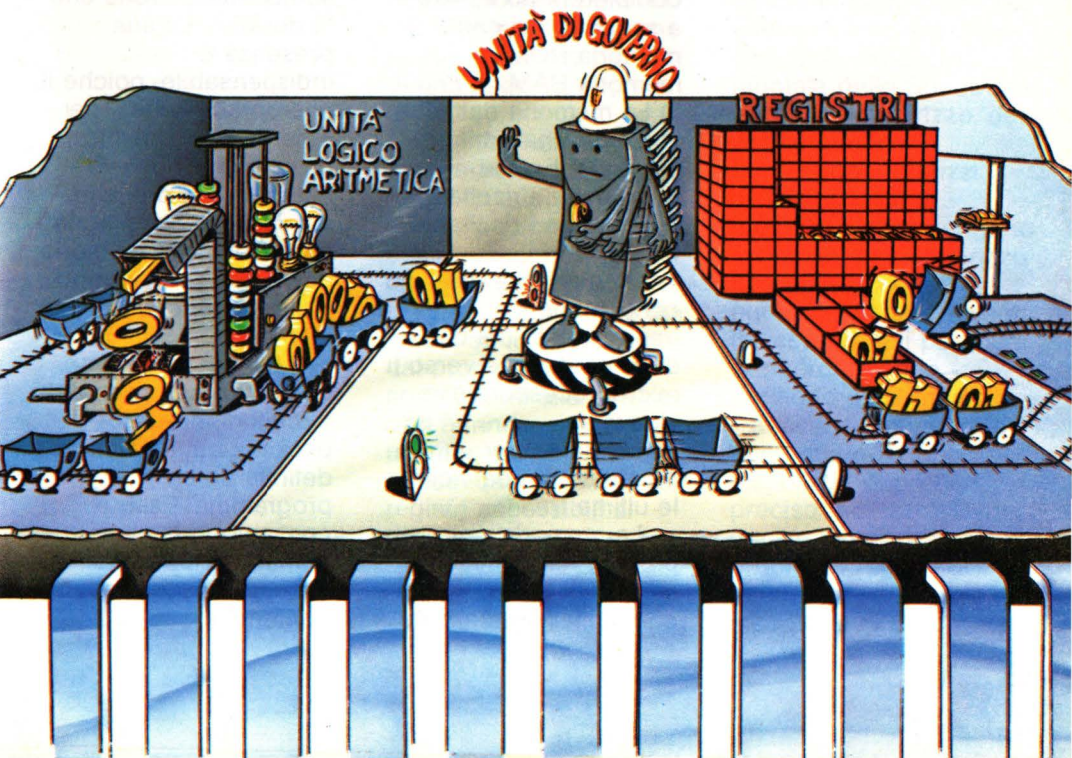
SOMMARIO

Introduzione

Prima di andare avanti .. facciamo un passo indietro. Per introdurre, infatti, il tanto famigerato linguaggio macchina (l. m. per chi vuol darsi un tono) e l'altrettanto noto assembler, occorre rinfrescare i concetti base nella CPU, la sua organizzazione interna e esterna, i bus.
Non è un'inutile esercitazione accademica; per ottenere mirabolanti prestazioni dal proprio computer è necessario - a volte indispensabile quando si vuole risparmiare occupazione di memoria e tempo di esecuzione - programmare direttamente il microprocessore che lo governa.

La CPU

Il termine CPU è spesso usato con due distinti significati. Considerando il computer completo di periferiche, si indica talvolta con CPU la parte contenente l'unità centrale propriamente detta (quella che esegue le istruzioni nella corretta sequenza), la memoria centrale e le interfacce di collegamento con il resto del calcolatore (e, in molti personal, anche la tastiera). In questo caso CPU (o unità centrale) è sinonimo di: "il computer vero e proprio, escluse le periferiche fisicamente separate (video, registratore, stampante, ecc..)".
Dal punto di vista logico, invece, la CPU è solamente l'unità centrale di calcolo e di controllo, escludendo quindi memorie ed interfacce di qualsiasi tipo. Visto che noi abbiamo sempre inteso il termine "CPU" (ed anche quello di "unità centrale") solo in questo secondo significato, continueremo a farlo anche nel resto dei nostri discorsi.
Fatta questa doverosa precisazione, entriamo subito nel vivo dell'argomento di questa lezione, cioè nella descrizione particolareggiata della CPU.
Finora abbiamo sempre accennato all'unità centrale come a una sorta di "scatola magica", alla quale rivolgevamo in ingresso delle domande (seguendo una determinata sintassi e grammatica) e da cui ottenevamo in uscita delle risposte. Adesso è giunto il momento di rimuovere questa limitazione, cercando di scavare un po' più in profondità. La conoscenza - più o meno approfondita - dell'unità centrale non è solitamente richiesta al programmatore; anzi, nei limiti del possibile, tutti i linguaggi ad alto livello (come il BASIC) cercano di evitare che tra hardware e software esistano legami di interdipendenza.
La fase di scrittura di un certo programma (per lo meno in un linguaggio ad alto livello) dovrebbe infatti essere sempre svolta in assoluta autonomia dalle caratteristiche hardware del computer utilizzato, in modo da risentire il meno possibile delle peculiarità costruttive

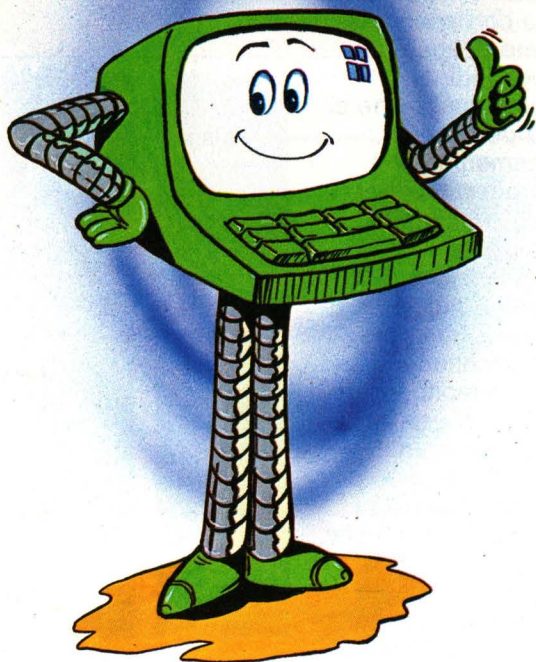
tipiche di un certo sistema.
Il BASIC, come ben sai - e soprattutto a causa di vari motivi storici che ne hanno provocato un'evoluzione anomala -, non può certo essere considerato il linguaggio in assoluto più "trasportabile" e "compatibile" (basta dare un'occhiata a uno stesso programma scritto per due diversi calcolatori per rendersene conto immediatamente!).
Negli ultimi anni sono stati tuttavia sviluppati numerosi altri linguaggi ad alto livello, che permettono di raggiungere l'obbiettivo della portabilità (cioè della quasi perfetta compatibilità tra una macchina e l'altra) in modo praticamente completo.
Tuttavia, in determinate circostanze, soprattutto nei casi di utilizzo particolarmente "evoluto" del computer o di programmazioni in linguaggi a basso livello (cioè molto più vicini alla macchina che all'uomo), può essere invece di estrema utilità fare specifico riferimento a un certo elaboratore e - di conseguenza - anche a uno specifico microprocessore. Oltre che a livello di cultura generale, la conoscenza della costituzione dell'unità centrale - per quanto semplificata e limitata - può pertanto risultare addirittura necessaria o addirittura indispensabile.

Organizzazione esterna di una CPU

Un microprocessore da solo non sa fare niente. Perché possa lavorare occorre collegargli delle memorie e delle interfacce. Una volta collegato a questi circuiti il microprocessore diventa allora un microcalcolatore, generalmente divisibile in quattro parti distinte, che comunicano le une con le altre tramite un bus dati, un bus indirizzi e un bus di controllo (per il momento puoi associare alla parola "bus" il corrispondente italiano di "collegamento"). Queste quattro parti sono:
1) Il microprocessore.
2) La memoria di programma, che segnala alla CPU le funzioni da compiere. Può essere - a seconda dei casi - memoria ROM o memoria RAM.
3) La memoria dati, che immagazzina i dati provenienti dalle periferiche, i risultati intermedi e i risultati finali.
4) Le interfacce, che consentono il trasferimento dei dati dalle periferiche verso ii microprocessore e viceversa.
Esaminiamo per un momento più da vicino le ultime tre.
— La memoria di programma contiene la sequenza di istruzioni che il microprocessore deve eseguire. Come già accennato, essa può essere sia di tipo RAM che ROM. Quando per esempio accendi il tuo computer la scritta che compare sullo schermo viene comandata alla CPU da un programma che si trova su ROM, mentre quando scrivi un'istruzione mediante tastiera essa viene memorizzata nella RAM.
— La memoria dati è invece un tipo di memoria notevolmente diverso dalla memoria del programma. È per forza di cose una memoria RAM, visto che deve poter essere letta o scritta tutte le volte che lo desideri. La sua presenza è indispensabile, poiché il microprocessore - per poter eseguire il suo programma - deve utilizzare dei dati che, per forza di cose, sono contenuti nella memoria dati o che provengono dalle periferiche tramite interfacce.
— Le interfacce infine, come ben sai, sono dei circuiti di ingresso-uscita la cui funzione è definita da un programma. Esse consentono la comunicazione tra il microprocessore e le periferiche.

Organizzazione interna di una CPU

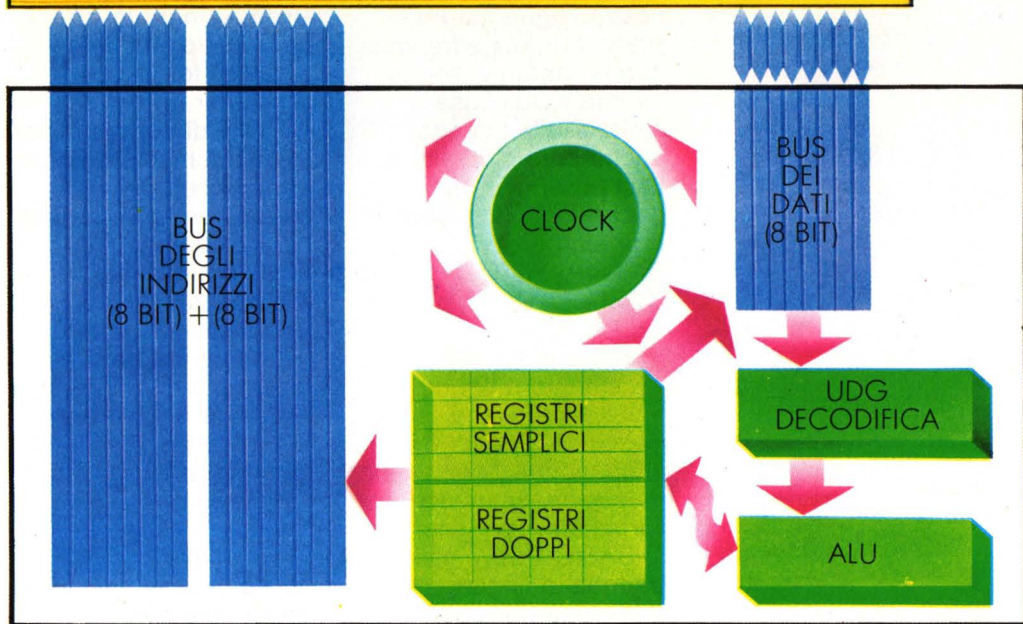
Un microprocessore contiene parecchie migliaia di transistor e di altri componenti elettronici: non è quindi il caso di descriverlo nei minimi particolari (e d'altra parte non ci sarebbe nemmeno di grande utilità).
Un microprocessore è comunque costituito da tre parti principali:
— l'unità di controllo
— l'unità aritmetico-logica
— i registri.
L'unità di controllo decodifica le istruzioni che vengono prelevate dalla memoria di programma ed elabora i segnali di comando necessari all'esecuzione di un'istruzione.
La funzione dell'unità aritmetico-logica (brevemente chiamata anche ALU) è invece l'esecuzione delle operazioni logiche ed aritmetiche sui dati che la alimentano attraverso le sue due porte di ingresso, che sono, rispettivamente, "l'ingresso destro" e "l'ingresso sinistro": queste porte si possono infatti immaginare come le due estremità più alte di una sagoma a forma di "V".
Dopo l'esecuzione di un'operazione aritmetica, come un'addizione o una sottrazione, l'ALU fa uscire i suoi contenuti dalla punta della "V".
I registri sono di due tipi: alcuni sono accessibili dal programma, altri sono interni alla CPU e non hanno un'importanza concettuale rilevante. I registri accessibili si dividono in tre categorie:
— i registri dei dati
— i registri degli indirizzi
— il contatore di programma (Program Counter), il puntatore dello stack e il registro di stato.
I registri dei dati sono in pratica delle locazioni di memoria che consentono la memorizzazione temporanea delle informazioni negli spostamenti tra l'unità aritmetico-logica, le memorie e le interfacce.
La loro lunghezza, cioè il numero di bit che compongono ciascun registro, è ovviamente uguale a quella della parola del microprocessore: 8 bit, se il microprocessore opera su 8 bit (come per esempio è il caso dello Z80, cioè della CPU montata nello Spectrum).
I registri di indirizzo, detti anche puntatori, contengono indirizzi delle posizioni di memoria i quali vengono inviati al bus di indirizzo con un'istruzione particolare che consenta l'accesso a tali posizioni.
Ti ricordi quando - parlando delle memorie - accennammo al fatto che ogni locazione disponeva di un ben preciso indirizzo? Bene, questo indirizzo serve proprio ai registri di indirizzo per fare in modo che la CPU possa

riferirsi a qualsiasi posizione della memoria. Il puntatore dello stack è un particolare registro di indirizzo che punta a una particolare zona della memoria, chiamata "area di stack". Viene automaticamente decrementato dopo ogni trasferimento di una informazione in questa zona, e incrementato dopo ogni prelievo. Il puntatore di stack, come abbiamo già avuto modo di accennare in una delle scorse lezioni, ha una funzione particolarmente importante nella chiamata e nel ritorno dai sottoprogrammi.
Il Program Counter (o contatore delle istruzioni) sorveglia l'esecuzione dell'intero programma. Viene caricato inizialmente con l'indirizzo della prima istruzione del programma, e in seguito segnala al microprocessore gli indirizzi delle istruzioni che devono essere eseguite successivamente. Mentre il microprocessore legge nella memoria di programma una istruzione, e la esegue, il contatore prosegue fino all'indirizzo dell'istruzione successiva e via di seguito. Il microprocessore in genere esegue le istruzioni in ordine sequenziale, cioè una dopo l'altra. Però, nel caso di alcune istruzioni l'esecuzione sequenziale dl programma può essere modificata: nel Program Counter viene allora caricato l'indirizzo di salto, e l'eventuale indirizzo di ritorno al programma principale viene conservato per essere utilizzato in seguito.
Il registro di stato contiene invece un certo numero di bit posti a 0 o a 1, a seconda che si siano verificate determinate condizioni dopo l'esecuzione di alcune istruzioni (per esempio, se il risultato è positivo o negativo, con riporto o senza, ecc.). Lo svolgimento del programma può essere modificato in funzione dei valori assunti da uno o più bit del registro di stato. È uno dei registri più importanti che il programmatore ha a propria disposizione. I registri possono infatti essere usati per varie verifiche speciali o condizioni eccezionali, oppure per verificare velocemente alcuni risultati errati.
Esiste infine un altro importantissimo registro che equipaggia la ALU: l'accumulatore. Questo è sempre uno dei due ingressi dell'ALU (poco importa se il "destro" o il "sinistro"); l'ALU fa infatti sempre automaticamente riferimento a questo accumulatore come ad uno degli ingressi.

Nelle operazioni aritmetiche e logiche, quindi, uno degli operandi sarà nell'accumulatore e l'altro si troverà tipicamente in una locazione della memoria. Il risultato sarà depositato nell'accumulatore. Il riferimento dell'accumulatore come sorgente e destinazione dei dati è la ragione del suo nome: esso accumula i risultati.
Il vantaggio di questo approccio basato sull'accumulatore è costituito dal fatto che si possono impiegare istruzioni molto più corte in linguaggio macchina. Se anche l'altro degli operandi dovesse infatti essere prelevato da uno degli altri registri (diversi dall'accumulatore), sarebbe necessario utilizzare istruzioni più lunghe per indicare l'indirizzo del registro. Perciò l'architettura dell'accumulatore si risolve in una maggiore velocità di esecuzione. Lo svantaggio è che l'accumulatore deve sempre essere caricato con i dati richiesti dall'operazione prima della sua utilizzazione. Questo può talvolta provocare qualche punto inefficiente.

I bus

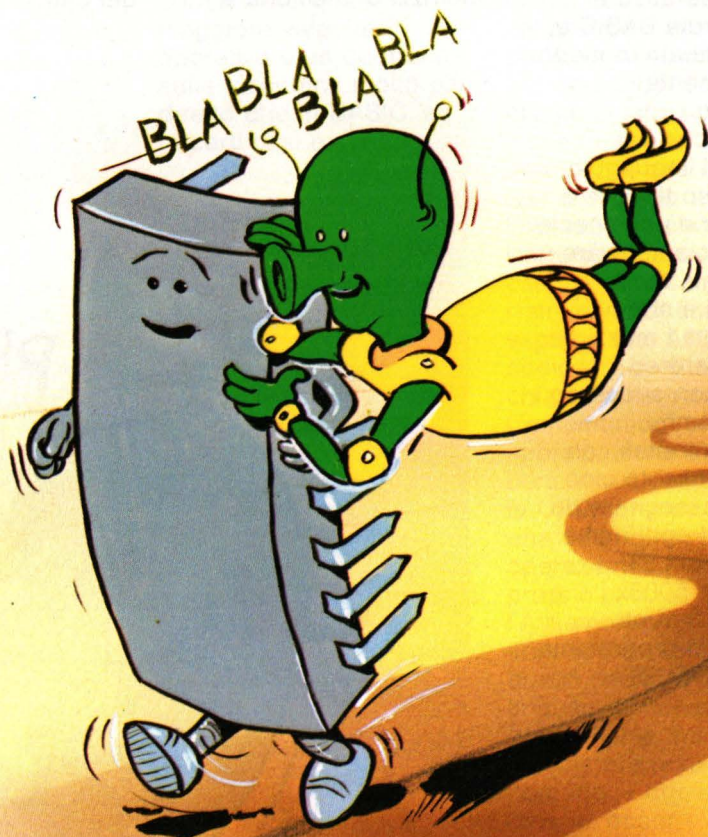
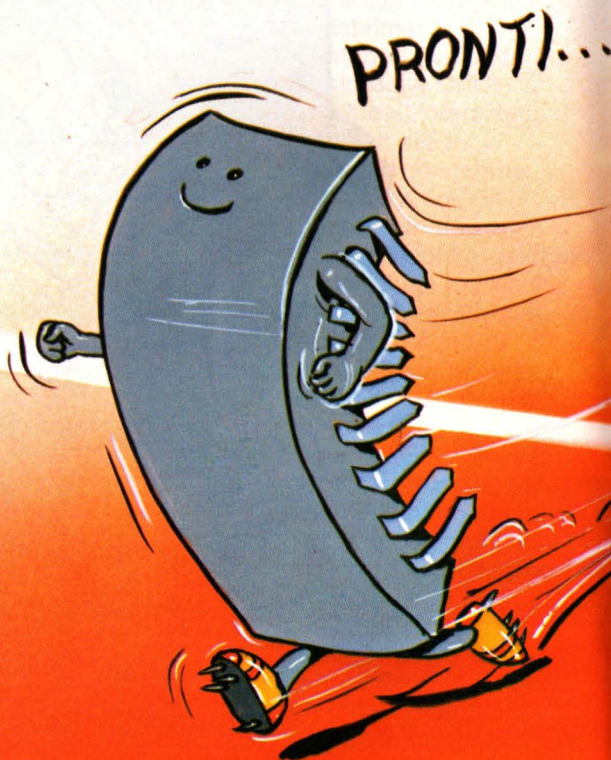
Il microprocessore comunica con le periferiche tramite tre bus:
— il bus degli indirizzi
— il bus dei dati
— il bus di controllo.
Il bus degli indirizzi è un insieme di linee elettriche, che consentono al microprocessore di selezionare una posizione di memoria o un registro di un'interfaccia. La CPU invia su queste linee, verso una periferica, un indirizzo codificato in binario. La periferica, ricevuto e decodificato l'indirizzo, seleziona il registro corrispondente.
Il numero di linee del bus degli indirizzi determina quella che si chiama potenza di indirizzamento del microprocessore; per esempio, 16 linee consentono di indirizzare 2 ↑ 16 (65536) locazioni di memoria. È da notare - tra l'altro - che il concetto di indirizzo è molto vicino a quello che si usa nel linguaggio corrente: la localizzazione di una persona in una città avviene in effetti tramite un indirizzo che contiene la strada ed il numero civico della strada.

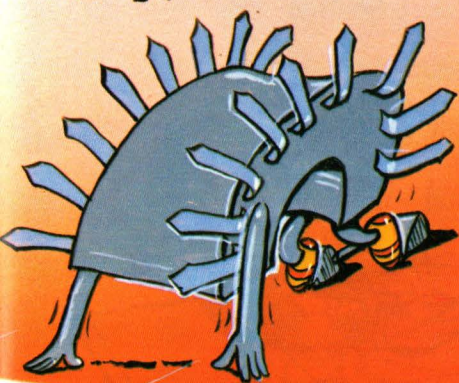
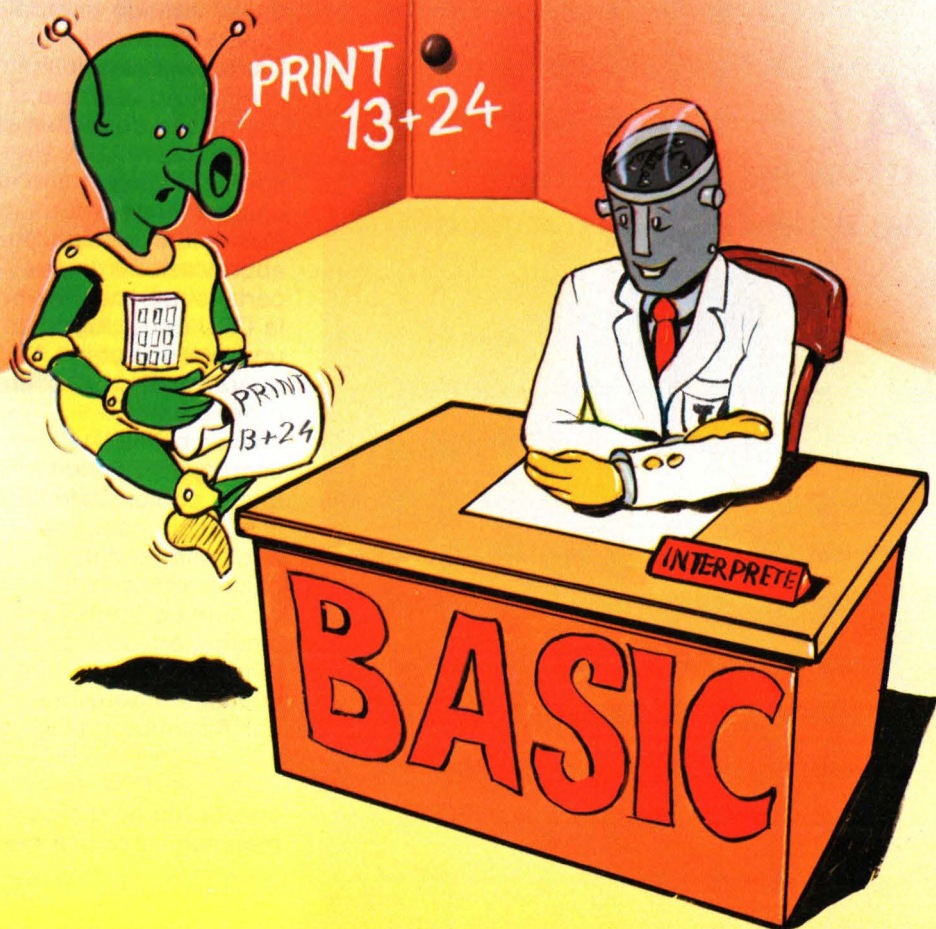
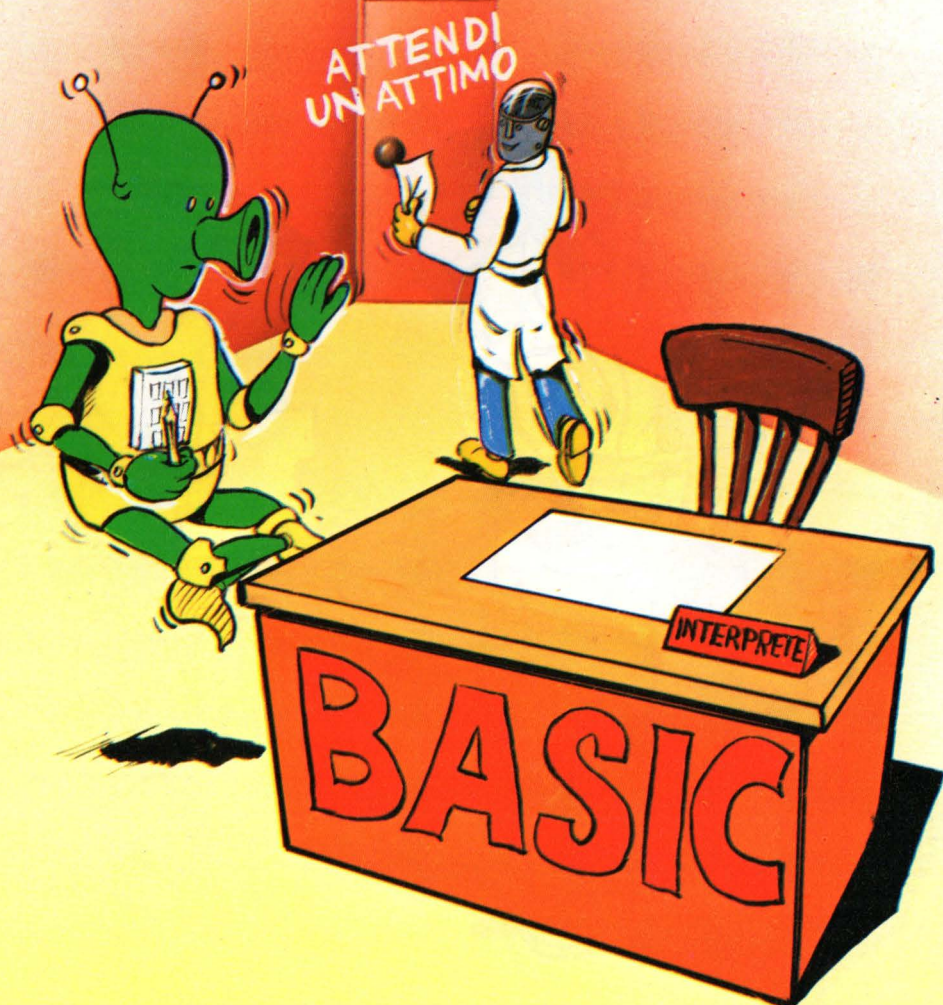
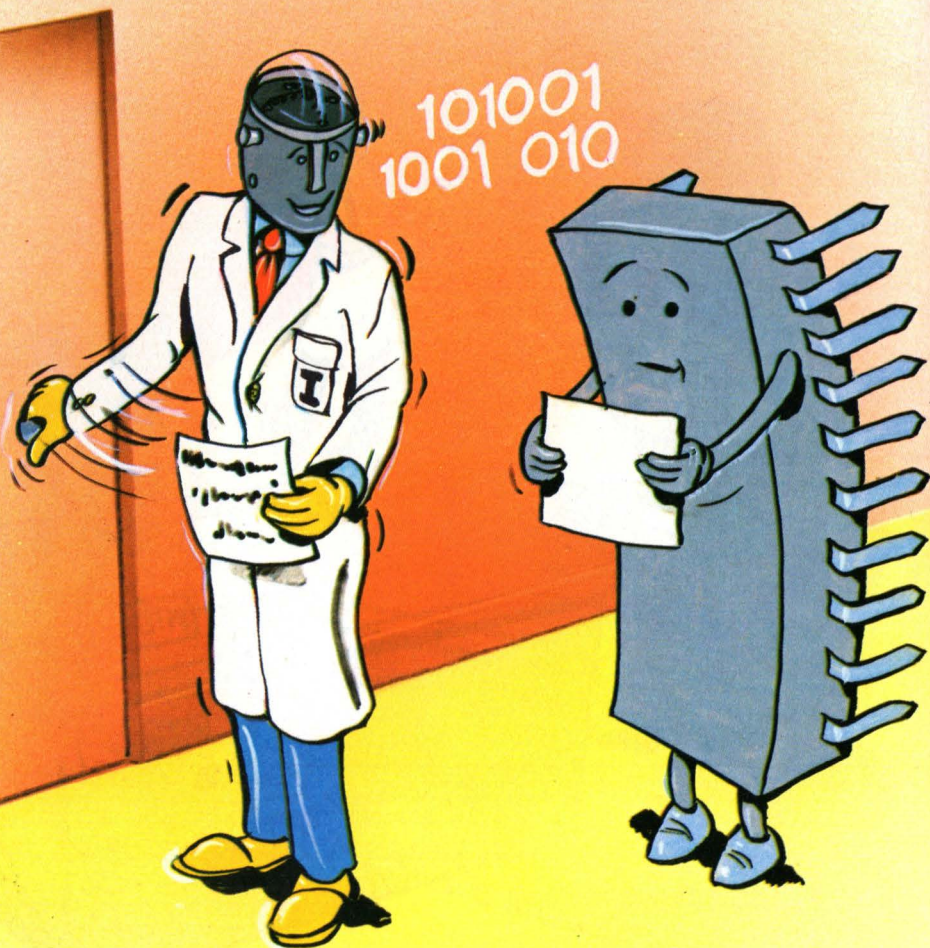
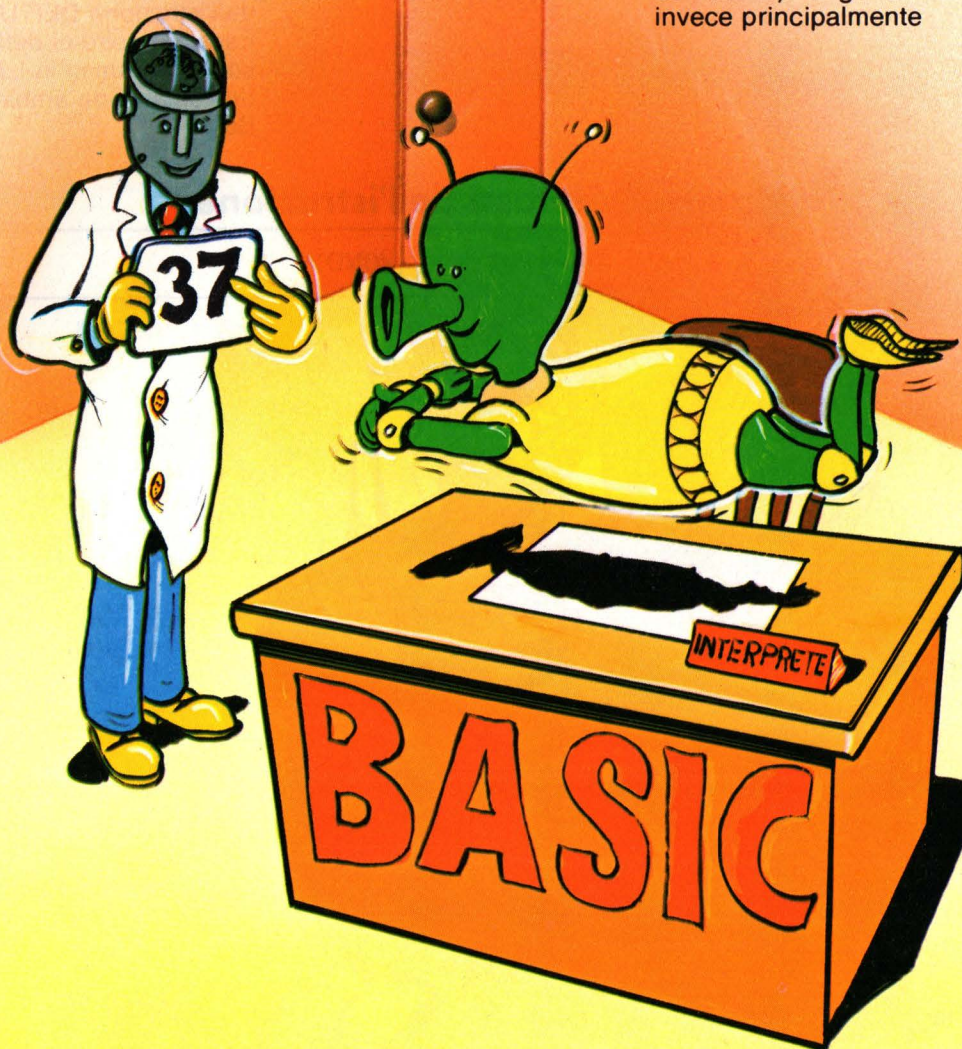
Il bus dei dati è anch'esso costituito da un gruppo di linee elettriche sulle quali avvengono gli scambi di dati tra il microprocessore e le periferiche (memoria ed interfacce). Il numero di linee di questo bus dipende dalla lunghezza della parola del

microprocessore: poiché il microprocessore dello Spectrum è a 8 bit, il bus dati dispone anch'esso di 8 linee.
Il bus di controllo è costituito da un certo numero di segnali di vario genere, che assicurano la sincronizzazione tra il microprocessore e le periferiche. Le funzioni più comuni assicurate da questo bus sono:
— la selezione di un'operazione di lettura o scrittura
— la interruzione del microprocessore (per interruzione si intende una procedura che - attraverso un segnale elettrico a un particolare piedino della CPU - segnala al microprocessore che una periferica vuole comunicare con lui)
— la richiesta di accesso al bus di una periferica
— il riconoscimento di una richiesta di accesso al bus
— altre funzioni meno comuni, ma altrettanto importanti di quelle viste finora.

Il linguaggio macchina

Abbiamo già detto varie volte che scrivere un programma significa scrivere una certa serie di istruzioni in un linguaggio comprensibile dal calcolatore, cioè in un linguaggio di programmazione.
Facciamo per un momento un'analogia con l'uomo. Noi possiamo parlare molte lingue (inglese, francese, tedesco, ecc.), possiamo comprenderle tutte, ma se ci mettiamo a fare un ragionamento o a pensare a un problema, ci accorgiamo che ne usiamo una sola. Questa lingua, usata dal nostro cervello, a volte è la nostra lingua madre - l'italiano -, ma altre volte non è che una "lingua interna". Qualcosa di molto simile accade anche per i calcolatori. Essi possono infatti capire molti linguaggi di programmazione, il BASIC, il FORTRAN, il COBOL, il PASCAL, ecc., ma nel loro interno ne usano uno ed uno soltanto, formato da lunghe file di zeri ed uno in codice binario. Questo linguaggio è quello della macchina. Qualunque istruzione - per esempio in BASIC - deve essere tradotta in linguaggio macchina per poter essere eseguita dal calcolatore. Tale compito nel BASIC del tuo Spectrum viene realizzato - ormai lo sai benissimo - dall'infaticabile interprete BASIC, vero e proprio traduttore simultaneo tra te e il tuo computer.
In alcuni casi può tuttavia essere utile (o addirittura indispensabile) ricorrere alla programmazione diretta del microprocessore, scavalcando i normali meccanismi (che inevitabilmente rallentano la velocità di calcolo della CPU) imposti dall'interprete.
Programmare in linguaggio macchina significa fornire al computer un certo insieme di configurazioni binarie (chiamate anche codici operativi) che il calcolatore è in grado di capire ed eseguire; ciascuna di esse rappresenta un'operazione eseguita via hardware dalla CPU, cioè mediante delle vere e proprie commutazioni di interruttori all'interno del computer.
La differenza fondamentale è che per programmare in BASIC non è necessario conoscere come

LINGUAGGIO

funziona il microprocessore con il quale è realizzato il calcolatore, mentre è necessario per programmare in Assembler (così viene normalmente chiamato il linguaggio macchina). In BASIC, a meno di

usare le istruzioni che leggono e scrivono direttamente nei byte di memoria (cioè PEEK e POKE) non devi mai occuparti di indirizzi di memoria: questo compito è affidato alle routine del sistema operativo e dell'interprete BASIC e viene effettuato in modo completamente automatico.
Una volta che hai imparato il linguaggio BASIC, esso ti mette a disposizione una specie di interfaccia software con la quale tu comunichi e che ti fornisce tutti i mezzi per programmare ed eseguire con successo i tuoi programmi.
I dati sono trattati con nomi simbolici (è cioè possibile assegnare un numero ad ogni nome: per esempio LET PREZZO = 5000); i riferimenti a punti specifici del programma (GOTO, GOSUB) si ottengono con riferimento ai numeri distintivi delle linee del programma BASIC.
In Assembler, invece, devi occuparti dei registri funzionali del microprocessore, degli indirizzi di memoria e ogni operazione deve essere pensata in tutti i suoi particolari, seguendo le caratteristiche del calcolatore. Inoltre, in Assembler risulta meno semplice il trattamento dei dati.

Svantaggi e vantaggi dell'Assembler

Il linguaggio macchina è in binario: puoi intuire come scrivere un programma in questo linguaggio sia lungo, laborioso e difficile.
D'altra parte i programmi in Assembler consentono una maggiore velocità operativa e un controllo sulla macchina molto più diretto che in BASIC, ed in certi casi è quindi necessario o conveniente sobbarcarsi a questa fatica, quando l'applicazione lo richiede.
Esistono diverse possibilità per scrivere programmi in linguaggio macchina, per esempio utilizzando - anziché un codice numerico - un codice simbolico, in cui ciascuna istruzione viene rappresentata da una parola che in qualche modo ricorda l'operazione che l'istruzione stessa esegue. Ad esempio, l'istruzione di somma si esprime con ADD. Un codice di questo tipo si chiama mnemonico ("mnemonico" significa abbreviazione di una certa istruzione, che per la CPU corrisponde ad una determinata operazione); per essere proprio precisi l'Assembler è il linguaggio costituito da questi codici. Dato che il linguaggio comprensibile direttamente dal calcolatore resta comunque quello binario, occorre disporre di un programma che effettui la traduzione dall'Assembler al linguaggio macchina vero e proprio (cioè sostituisca al codice mnemonico di una certa

istruzione il corrispondente codice numerico); questo programma prende il nome di assemblatore. Esistono in commercio numerosi assemblatori: il loro utilizzo, per chi desidera programmare seriamente in Assembler, è praticamente indispensabile. Tenuto conto del carattere introduttivo al linguaggio macchina di questa e delle prossime lezioni, non è comunque assolutamente necessario che ti premuri di procurartene uno.
Per concludere, queste sono le principali cause che possono giustificare

l'impiego dell'Assembler:
— tutte le volte in cui il fattore tempo gioca un ruolo fondamentale. Tra il tempo di esecuzione di un programma che funziona sotto l'interprete BASIC e quello di un programma scritto direttamente in Assembler può esserci un enorme differenza di velocità (il linguaggio macchina può essere fino a 2-300 volte più veloce);
— tutte le volte in cui il fattore spazio è di essenziale importanza. La dimensione di un programma BASIC, aggiunta a quella del suo interprete, sarà sempre superiore a

quella dello stesso programma (che esegue la stessa funzione), realizzato in Assembler. Tale fattore, con il progressivo ed inarrestabile calare dei prezzi delle memorie, sta comunque diventando sempre meno importante;
— tutte le volte in cui è necessario realizzare delle istruzioni sconosciute all'interprete BASIC, o impossibili da realizzare con un linguaggio ad alto livello (come appunto il BASIC).
Dato che un programma in Assembler risulta molto più veloce di un programma BASIC, ma ci vuole molto più tempo a scriverlo e risulta più difficile trovarne gli errori, sarà sempre necessario prendere di volta in volta una

decisione sulla strada che conviene scegliere, valutando bene le diverse esigenze, le possibili alternative ed i rispettivi costi (oltre che in termini di tempo, anche di fatica). Inoltre, devi tener presente che certe particolari applicazioni, come l'utilizzo del computer per giochi di animazione, si possono realizzare bene solo programmando in Assembler. In altri casi (esempio tipico: i programmi di gestione aziendale o di contabilità) vengono invece principalmente

richieste doti di leggibilità e di modificabilità dei programmi, ottenibili esclusivamente utilizzando un linguaggio ad alto livello.
Naturalmente, è sempre possibile (e lo si fa spesso) conciliare le due forme di linguaggio, scrivendo ad esempio l'ossatura del programma in BASIC e ricorrendo, quando è necessario, a dei piccoli programmi realizzati in Assembler. Esamineremo tra poco questa possibilità.

DEF USR e USR

Come è possibile infatti avvantaggiarsi della flessibilità del linguaggio macchina senza rinunciare alla comodità e potenza del Basic? Semplicemente utilizzandoli insieme! Ciò è reso possibile dall'istruzione DEF USR. Essa permette di definire sottoprogrammi in L.M. richiamabili da ambiente Basic.

Sintassi dell'istruzione

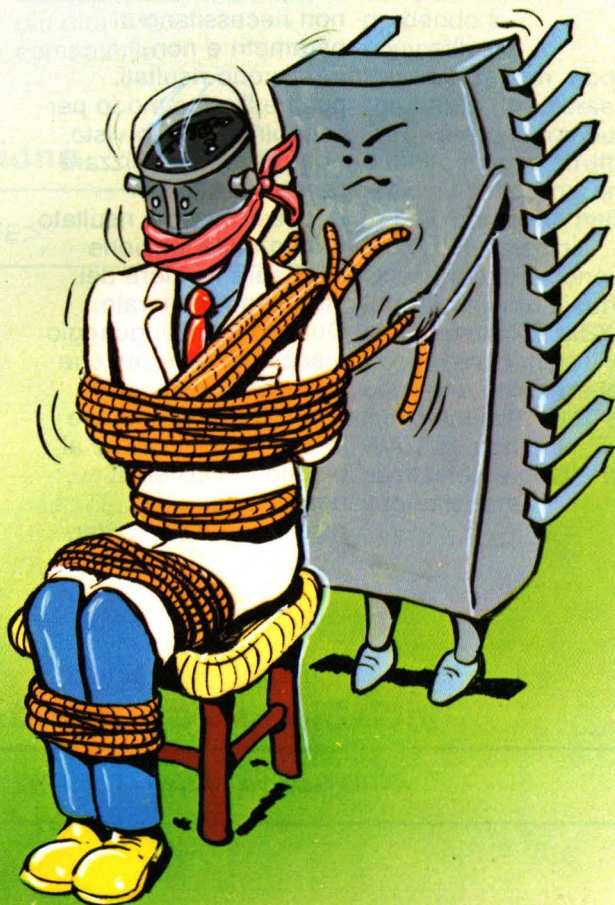
DEF USR<N> = <IND>

LINGUAGGIO

<N> è un numero intero compreso tra 0 e 9, che permette di definire fino a 10 sottoprogrammi diversi.

<IND> è l'indirizzo di inizio del sottoprogramma,cioè la posizione nella memoria della prima istruzione in L.M. da eseguire.

Una volta definiti, i sottoprogrammi in linguaggio macchina si richiamano come una funzione del Basic, semplicemente con

USR<N> (<PARAMETRO>)

dove <N> è il numero di riconoscimento della funzione, e <PARAMETRO> può essere una variabile Basic di qualsiasi tipo, numero o stringa, sulla quale potrà agire il sottoprogramma in L.M. Ad esempio:

DEF USR5 = 48000 definisce un sottopogramma in linguaggio macchina con inizio all'indirizzo 48000.

```
PRINT USR2
```

visualizza il risultato fornito dal sottoprogramma L.M. numero 2 con parametro di ingresso D.
Molto spesso i sottoprogrammi in L.M. non necessitano di parametri e non forniscono risultati; poiché l'unico modo per eseguirli è quello visto, non resta che utilizzare parametri fittizi.
In questo caso il risultato fornito dalla funzione USR sarà il valore del parametro inalterato.
Quando usi il linguaggio macchina, ricordati che un errore può essere ben più disastroso che in Basic; può portare al blocco del sistema, riparabile solo resettando il computer.

VARPTR

Questa funzione serve a conoscere l'indirizzo al quale è memorizzato il primo byte dei dati associati ad una variabile, un array o al blocco di controllo di un file.
Il risultato di questa funzione è un intero compreso tra —32768 e 32767.
Per ottenere un risultato corretto in caso di segno negativo, basterà sommare 65536.

PRINT HEX$(VARPTR(I%))
Stampa il valore esadecimale dell'indirizzo del primo byte della variabile I%.

POKE VARPTR(A),0
Azzera il primo byte della variabile A.

U=USR1(VARPTR(Z(0)))
Assegna ad U il risultato del sottoprogramma in L.M. numero 1 a cui è stato passato come parametro l'indirizzo di inizio dell'array Z.

Sintassi della funzione

VARPTR (VAR)

CALL

L'istruzione CALL (abbreviabile con il carattere "—") serve a richiamare un'estensione del Basic apportata da una cartuccia ROM. Naturalmente il programma che fa uso di CALL potrà funzionare correttamente solo in presenza dell'apposita cartuccia, e, di conseguenza, non potrà più dirsi MSX compatibile.

Sintassi dell'istruzione

CALL <NOME ESTENSIONE>

INP, OUT, WAIT

Queste istruzioni servono a leggere, scrivere, o attendere valori sulle porte di I/O. Dato che gli indirizzi utilizzati per le porte "vitali" dei computer MSX possono variare da modello a modello, un programma che fa uso di tali istruzioni può non funzionare su tutti i computer MSX, perdendo la compatibilità.
Per utilizzare in modo "originale" le risorse hardware del computer, è molto meglio sfruttare appositi sottoprogrammi in L.M. contenuti nella ROM del sistema. Questi garantiscono un funzionamento sicuro e la compatibilità MSX.
In ogni caso, non ti conviene utilizzare le istruzioni INP, OUT e WAIT se non sai esattamente quello che vuoi ottenere.

Sintassi delle istruzioni

INP, OUT, WAIT NUM PORTA

Alla scoperta dell'Assembler

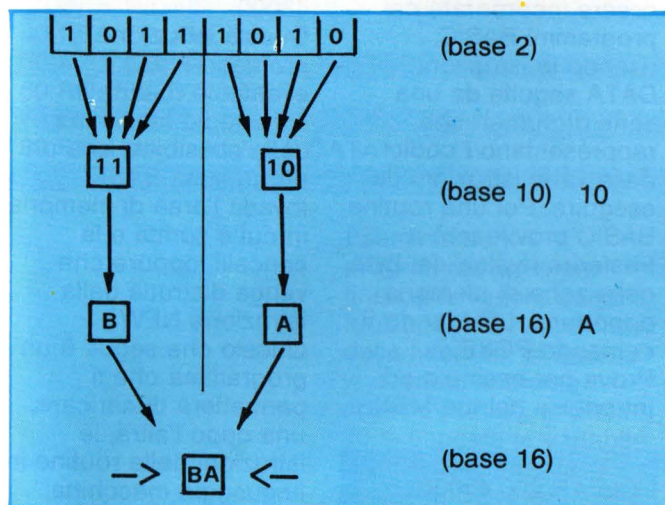
Prima di passare direttamente alla scrittura di programmi in linguaggio macchina, dobbiamo ancora affrontare importanti (anzi, indispensabili) argomenti, grazie ai quali potremo successivamente avanzare nei nostri discorsi in modo più semplice e spedito.
Ti potrà forse sembrare che il linguaggio macchina non faccia per te: troppe cose da imparare, da sapere e da tenere a mente.
Questo può anche essere vero: tuttavia una volta imparati i concetti fondamentali, molti argomenti ti diventeranno molto meno complicati di quello che potevano apparire a prima vista.
Inoltre più ti addentrerai nell'assembler - diventando di conseguenza più padrone della situazione - più i risultati diventeranno spettacolari e carichi di soddisfazioni.

La numerazione esadecimale

Il primo, importante punto è determinare un sistema di numerazione che giunga a un compromesso tra la difficoltà che l'uomo incontra nel leggere e lavorare con i numeri binari e l'avversità che la CPU possiede verso le cifre decimali.
Ecco quindi che interviene un terzo e fondamentale (per chi programma in Assembler) sistema di numerazione, cioè quello in base 16. Con questa base un numero binario a otto cifre può essere scritto con due cifre in base sedici, cioè con due numeri esadecimali.
Ricorderai certamente che un numero in base due è composto solo da cifre 0 e 1, mentre uno in base 10 è scritto con cifre comprese tra 0 e 9; così un numero in base 16 dovrà essere scritto con cifre comprese tra 0 e 15. Però, dal momento che non abbiamo cifre maggiori di 9, usiamo le prime sei lettere dell'alfabeto:

10 = A
11 = B
12 = C
13 = D
14 = E
15 = F

Il modo più semplice di convertire un numero binario a 8 bit in un numero esadecimale è quello di separare gli 8 bit in due gruppi di 4 bit, effettuare la conversione da una base all'altra di ciascun gruppo e poi ricombinare il risultato.

Per esempio:

Con questo metodo possiamo convertire qualunque valore di un byte in due cifre esadecimali. Il vantaggio è quello di avere una scrittura più compatta della notazione binaria e, con un minimo di allenamento, quasi identica al sistema decimale. In ogni caso, tutti questi sistemi di numerazione sono - ricordiamolo - modi diversi di rappresentare gli stessi numeri. Tutto sta nel conoscere la base utilizzata. Per convenzione, allo scopo di riconoscere immediatamente un numero esadecimale, scriveremo sempre i numeri in base 16 con il suffisso H. Così

```
10
```

significherà "dieci decimale", mentre

```
10H
```

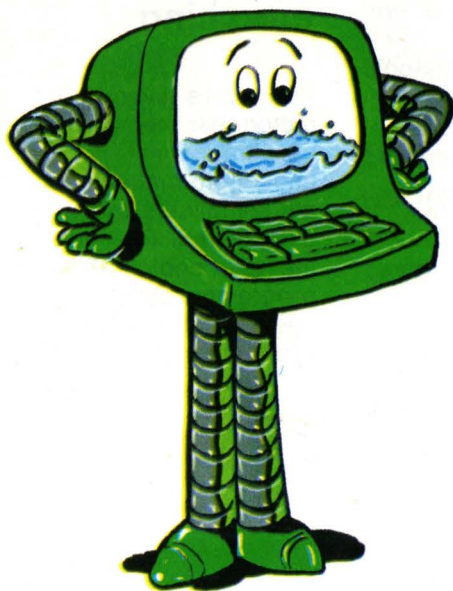
significherà "uno zero esadecimale" (cioè 16 decimale).

Come memorizzare i programmi in linguaggio macchina

Puoi usare diversi metodi per memorizzare i programmi in linguaggio macchina. Il più semplice è quello di abbassare il TOP della memoria dedicata al programma BASIC, agendo sul puntatore RAMTOP.

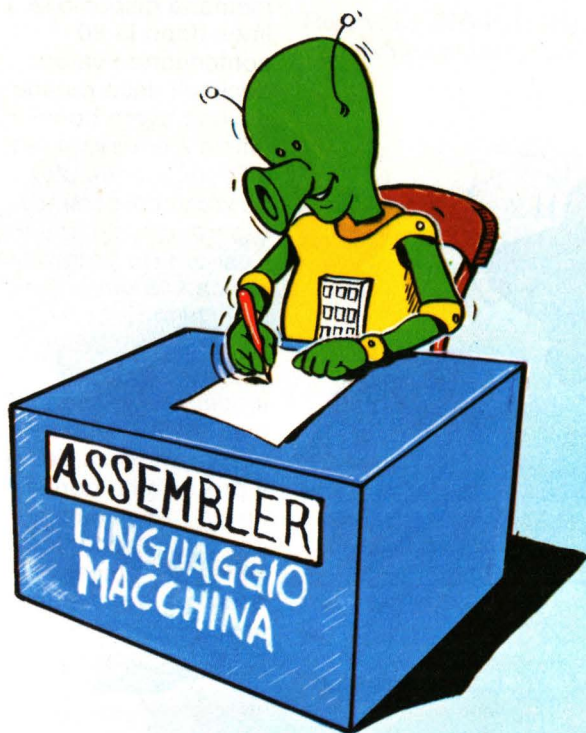
L'abbassamento del TOP della memoria si ottiene con il comando CLEAR seguito dal numero di byte da riservare alle variabili e dall'indirizzo dell'ultimo byte che vuoi avere disponibile per il programma BASIC.
Le routine in linguaggio macchina possono essere incorporate nei programmi BASIC, usando le istruzioni DATA seguite da una serie di numeri che rappresentano i codici delle varie istruzioni da eseguire. Poi una routine BASIC provvederà a trasferire i valori dei byte nella zona di memoria opportuna, utilizzando il comando POKE.
Prova per esempio ad introdurre nel tuo MSX:

```
CLEAR 200,48000
```

Riserverai 200 byte alle variabili e limiterai l'area Basic entro l'indirizzo 48000.
Questa istruzione proteggerà una eventuale routine in linguaggio macchina dalla possibilità che un programma BASIC invada l'area di memoria in cui è scritta e la cancelli, oppure che venga distrutta dalla istruzione NEW.
Questo che segue è un programma che ti permetterà di caricare, una dopo l'altra, le istruzioni delle routine in linguaggio macchina:

```
10 CLEAR 200,48000!
20 IND=48000!
30 IND=IND+1:
40 READ CODICE
50 IF CODICE=— 1 THEN END
60 POKE IND,CODICE
70 GOTO 30
80 REM seguono linee DATA
```

Esaminiamo il programma linea per linea, per vedere come funziona:
della linea 10, di quanto vi accade, abbiamo appena parlato; in linea 20 si imposta la variabile IND(IRIZZO) che verrà via via aggiornato ad ogni successivo passaggio per la linea 30.
Da quest'ultima, infatti ha inizio il ciclo all'interno del quale si procede alla lettura del dato (linea 40), al controllo che non si tratti dell'ultimo (linea 50), alla sua memorizzazione all'indirizzo puntato (LINEA 60). La linea 70 chiude il ciclo.
Naturalmente, non è necessario avere tutti i valori su un'unica linea DATA; puoi usare quante linee desideri, compatibilmente con la memoria disponibile. Le linee dopo la 80 contengono i valori decimali della routine e devono avere come ultimo elemento il —1, affinché il computer possa accorgersi di essere arrivato in fondo.
Così questo programma carica il codice macchina nella RAM.
Per far eseguire la routine occorrerà modificare la linea 50, in modo tale che in corrispondenza del — 1 il programma non si arresti, ma vada a comandare una linea DEF USR. Questo esercizio (tutto sommato veramente elementare) lo lasciamo fare a te.

Esempi Assembler

Proviamo adesso a scrivere qualche programma molto semplice in Assembler, illustrandone contemporaneamente il funzionamento e il confronto con il corrispondente listato BASIC. Non sempre è possibile fare tale paragone: gli esempi che tratteremo ci daranno tuttavia questa possibilità.
Come primo esempio proponiamoci di trasferire il contenuto di due locazioni di memoria (per esempio la 5428 e la 5429) in un'altra coppia di locazioni (scegliamo la 52000 e la 52001). In BASIC scriveremmo:

```
10 POKE 52000, PEEK 5428
20 POKE 52001, PEEK 5429
```

e tutto sarebbe risolto. Vediamo adesso come fare in Assembler. Ecco qui il listato:

```
LD HL,(1534H)
LD (CB20H), HL
RET
```

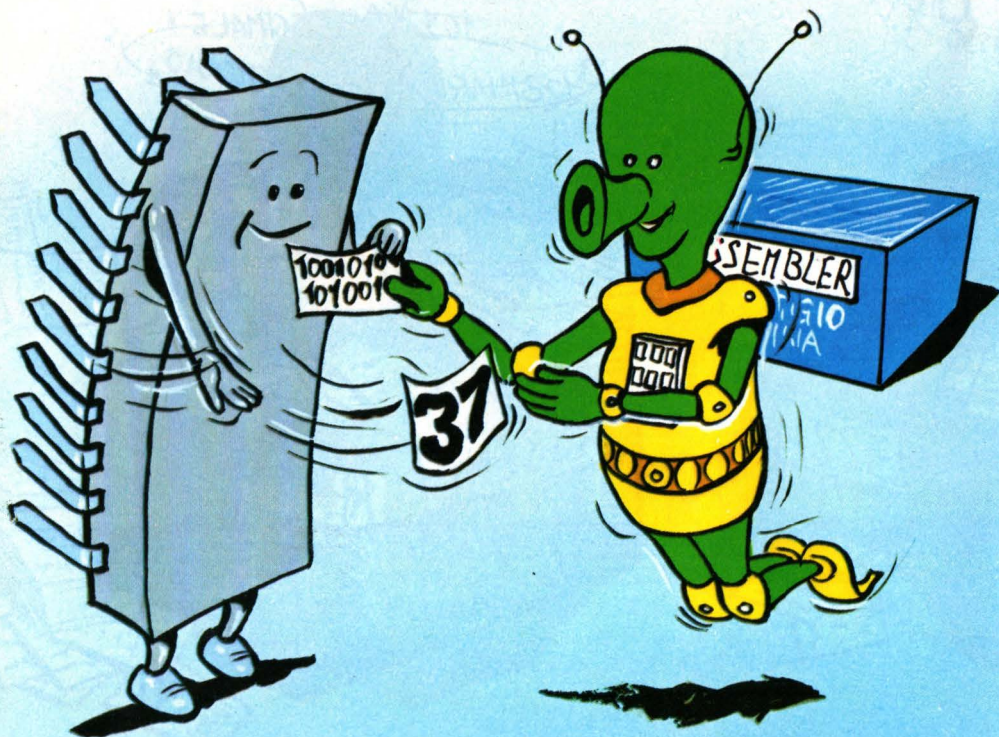
La prima riga carica (LD è abbreviazione di LOAD, cioè carica) il registro HL con il numero contenuto nella coppia di locazioni poste a partire dall'indirizzo 5428 (1534 esadecimale). La seconda memorizza nella coppia di locazioni CB20 e CB21 (52000 e 52001 decimali) il valore contenuto in HL. La terza riga è il famoso RETURN, che serve per restituire il controllo al BASIC.
È interessante osservare che abbiamo in pratica lavorato su 16 bit alla

volta, trasferendoli prima in un registro doppio (HL indica che abbiamo utilizzato nello stesso momento i registri H e L) e quindi in due locazioni da 8 bit. Questo fatto è permesso dalla particolare progettazione dei registri dello Z80, che, pur essendo a 8 bit, possono essere composti in modo da ottenere registri a 16 bit.

Notiamo ora la costituzione delle istruzioni: un operatore e uno o più operandi. In LD HL,1534H - per esempio - LD è l'operatore, mentre HL e 1534H sono gli operandi. È necessario fare una precisazione sulla parte (o, meglio, sul campo) relativa agli operandi. Questo campo dipende strettamente dal tipo di operazione, ed è sempre appartenente a uno dei seguenti gruppi:

— a un solo argomento
— a due argomenti, separati da un carattere speciale (generalmente la virgola). In quest'ultimo caso il primo argomento rappresenta sempre la destinazione e il secondo la sorgente.

Così, in LD HL,(1534H), la sorgente (cioè il numero contenuto in 1534H) deve andare in HL, che è la destinazione. La stessa cosa accade in LD (CB20), HL.

Passiamo ora alla conversione del programma nei codici

numerici eseguibili (questi che abbiamo appena visto sono solo i mnemonici). Occorre prendere in mano la tabella fornita dal costruttore del microprocessore, e convertire uno ad uno i vari termini. A ciascun elemento corrisponderà naturalmente un preciso codice numerico:

```
LD HL, (1534H)           2A 34 15
LD (CB20), HL            22 20 CB
RET                   —> C9
```

I numeri che abbiamo scritto sono ancora in esadecimale: prima di essere scritti nelle DATA dovranno essere convertiti in decimali.
L'ultima cosa da notare è che - se guardi bene - gli indirizzi delle locazioni sono stati scritti ribaltati (cioè 1534 è stato scritto 3415, e analogamente per CB20). Ciò è dovuto a ragioni costruttive del microprocessore, il quale pretende che di qualsiasi indirizzo gli vengano forniti prima il byte basso e poi quello alto.
Alla fine otteniamo i seguenti valori:

```
2AH = 42
34H = 52
15H = 21
22H = 34
20H = 32
CBH = 203
C9H = 201
```

che, introdotti nella linea DATA (ricordandosi di aggiungere anche il 999), formeranno la routine.
Esegui il programma e comanda quindi una USR. Prova quindi a verificare con delle PEEK nelle locazioni 52000 e 52001 - che effettivamente il lavoro sia stato compiuto!

Proviamo ora questo secondo esempio:

```
LD HL,(F7F8)     ;carica in HL il contenuto della
                  coppia di byte di
                  indirizzo 52000 e 52001
INC HL           ;lo incrementa di 1
INC HL           ;ancora
INC HL           ;ancora
LD (F7F8), HL    ;carica in 52000 e 52001
                  il contenuto di HL
RET              ;ritorna al BASIC.
```

Da notare che in Assembler i commenti vengono separati dalle istruzioni mediante un punto e virgola.
L'equivalente BASIC del programma appena visto sarebbe:

```
10 X=PEEK(52001!)+256*PEEK(52000!)
20 X=X+1
30 X=X+1
40 X=X+1
50 POKE52000!,INT(X/256):POKE52001!,
   X—256*PEEK(52000!)
60 RETURN
```

La conversione del programma nei codici numerici porta ai seguenti valori:

```
42, 248, 247, 35, 35, 35, 34, 248, 247, 201
```

Se eseguirai questa routine mediante una PRINT USR (P) vedrai che sullo schermo apparirà P+3. Ecco infine il listato Basic che compendia i punti sin qui svolti.

Conversione di un carattere in LM

Al di là della reale possibilità di utilizzo, l'obiettivo importante di questa proposta è quello di sottolineare il corretto procedimento da seguire per impostare ed eseguire un qualsiasi programma in linguaggio macchina.
Per sommare 3 ad un numero puoi usare benissimo il BASIC, ma l'esempio ci dà l'opportunità di evidenziare le 3 fasi fondamentali necessarie ad introdurre e a utilizzare routine in LM nei tuoi programmi.

PROGRAMMAZIONE

1 **Preparazione:**
Si fissa la RAMTOP affinché il programma BASIC non possa in alcun modo sovrapporsi al codice macchina.

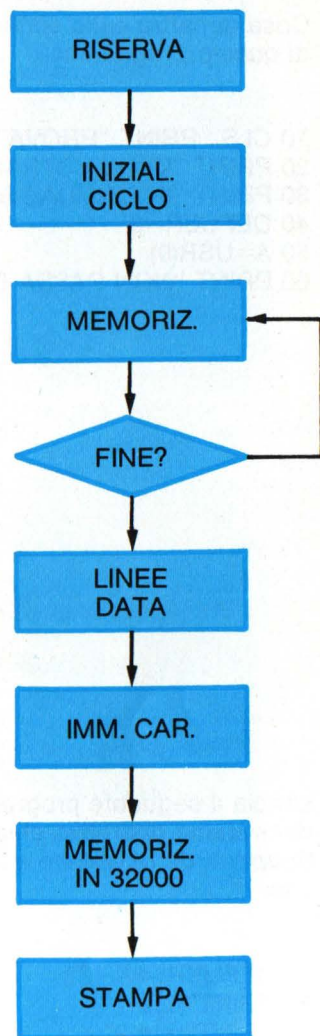
2 **Memorizzazione:**
si introduce in memoria il codice macchina.

3 **Esecuzione:**
si lancia il programma in LM.

PREPARAZIONE

MEMORIZZAZIONE

ESECUZIONE

```
10 CLEAR 200,48000!
20 IND=48000!
30 IND=IND+1
40 READ CODICE
50 IFCODICE=—1THEN100
60 POKEIND,CODICE
70 GOTO 30
80 DATA 42, 248, 247, 35, 35, 35, 34, 248, 247, 201, —1
90 REM esegue lm
100 DEFUSR1=48001!
110 REM passa parametro cui sommare 3
120 INPUT "numero =";N%
130 PRINT USR1(N%)
140 IFN=—1 THEN END ELSE 120
```

VIDEOESERCIZI

Cosa apparirà sullo schermo al termine dell'esecuzione di questo programma?

```
10 CLS : PRINT "PROVA"
20 PRINT "DI UNA STRANA ROUTINE"
30 PRINT "IN LINGUAGGIO MACCHINA"
40 DEFUSR=0
50 A=USR(0)
60 PRINT "NON PASSA DI QUI"
```

Lancia il seguente programma e, tramite esso la routine del sistema operativo chiamata dalla locazione 195. Scoprine da te l'effetto e l'istruzione Basic che ne fa uso.

```
10 DEFUSR7=&HC3
20 PRINT"riga 1"
30 PRINT"riga 2"
40 PRINT"riga 3"
50 PRINT"riga 4"
60 PRINT"riga 5"
70 Q=USR7(X)
80 PRINT "dove si vedrà
   questa scritta?"
```